



V x W o r k s 5 . 1 . 1

BSP Manual for EUROCOM-17

Revision 1 A

NAME

Eltec EUROCOM-17

INTRODUCTION

This manual entry provides board-specific information necessary to run VxWorks. Before running VxWorks, verify that the board runs in the factory configuration using vendor-supplied monitor program (RMon) and jumper settings, and check operation through the RS-232 connection.

BOOT ROMS

The EUROCOM-17 uses a flash EPROM for the resident monitor program (RMon) which is always resident on the EUROCOM-17. The RMon performs various initialization functions on the board and then optionally transfers to the program resident within the user EPROM. The user EPROM, located at U1602, is used to program VxWorks. This EPROM must be a 32 pin device and can be from 128 Kbyte to 1 Mbyte (27C010 to 27C080).

The Rmon needs to be set up such that the RMon automatically starts execution of the program resident in the user EPROM memory. To set up the automatic start of the program in user EPROM, set rotary switch number 2 to position 8 or higher. When the switch is set in this manner, the RMon will transfer control to the program located in the user EPROM.

The EUROCOM-17 has non-volatile RAM; thus, boot parameters are preserved whenever the system is powered off.

To load VxWorks, and for more information, follow the instructions in the **Getting Started** chapter of the **VxWorks Programmer's Guide**.

JUMPERS

The EUROCOM-17 factory configured for immediate operation with VxWorks. The table below lists the jumpers for the EUROCOM-17. Starred settings (*) indicate the factory default and are appropriate for use with VxWorks.

Jumper	Description
J1401	Watch dog period (closed * when 100 ms.) (open when 1.6 seconds)
J1601	Flash programming voltage (open * when not present) (closed when present)
J1605	Pin1 connection for EPROM (1-2 * when 5 volts) (2-3 when A19)

SWITCHES

The EUROCOM-17 has two rotary switches on the front panel of the processor board. Switch number 1 is used to select the VMEbus address (A32 and A16) for RMon. The addressing is as defined in the following table.

Position	A32 Space	A16 Space
F	0xF0000000	0xF000
E	0xF0000000	0xF000
D	0xF0000000	0xF000
2	0x20000000	0x2000
1	0x10000000	0x1000
0	RMon defined	RMon defined

A24 addressing is always disabled by using the RMon defaults.

VxWorks can inherit the addressing assignments set up by RMon, if desired. When VxWorks is built with the USE_RMON_CONFIG defined, then VxWorks will use the RMon addressing. When USE_RMON_CONFIG is not defined, the addressing is set as per the LOCAL_MEM_BUS_ADRS definition within config.h and the processor number.

Switch number 2 is used to select the start up mode that is used by RMon. For automatic transfer to the the VxWorks programmed in the user EPROM, this switch must be set to positions 8 - F. All other positions will not perform an automatic start of VxWorks.

Switch S3 on the front panel is used to select VME system controller functions. When the EUROCOM-17 is placed in slot 1 of the VME chassis, the system controller option should be selected. When another processor is placed in slot 1, then that processor will usually fulfill the system controller function and th S3 should not select the system controller function.

DEVICES

The cd2400Serial.c Cirrus Logic CD-2401 tty driver is provided for the four on-board serial ports; see the manual entry for **tyCoDrv**.

The chip drivers included are:

nvRam.c - non-volatile RAM

z8536Timer.c - Zilog Counter/Timer Parallel I/O (CIO) timer

vic064Vme.c - Cypress VIC-64 VME interface

A network interface for the on-board ILACC chip allows VxWorks to run without an additional network board. The interface is called "il" and should be specified as the boot device to the boot ROMs.

A device driver for the NCR 53C720 SCSI Input Output Processor (SIOP) is provided. The INCLUDE_SCSI directive must be enabled in config.h.

SPECIAL CONSIDERATIONS

The ILACC chip needs to be informed of its Ethernet address before it can attach to the LAN. The address is obtained from the RMon EPROM which is unique for each board.

Console connection is through serial port number 1 which is located on the the front connector. An adapter cable is available for this interface. The adapter cable converts the telephone jack connector to a D9 connection in the DCE configuration.

The NCR53C520 SCSI library supports SCSI wide and SCSI fast, however, these features are currently not supported by the SCSI library. Therefore, special functions are present to place a compatible SCSI device in the fast and/or wide mode. See the manual pages for **sysScsiFast** and **sysScsiWide** for **additional information**.

The EUROCOM-17 has a watchdog function that when enabled must be triggered at a minimum of the frequency selected by the J1401 setting (100 ms. or 1.6 seconds). The INCLUDE_WDOG definition within config.h enables this feature.

The VIC-64 VME interface driver provides an optional VME DMA copy function which is enabled by the inclusion of the INCLUDE_VIC_BLK_XFER definition within the config.h file. See the manual entry for **sysVicBlkCopy** for additional information on block copies.

BOARD LAYOUT

The EUROCOM-17 does not have any jumpers that need be in set positions for operation with VxWorks. Refer to the EUROCOM-17 technical manual for the locations of jumpers, switches, and EPROMs.

SEE ALSO

Programmer's Guide: Getting Started, Configuration RMon Monitor Program - Programmers Reference Manual , RMon User Manual - Software Manual , EUROCOM-17 Technical Manual

NAME

sysLib - Eltec EUROCOM-17 system-dependent library

SYNOPSIS

sysNvRamGet() - get the contents of non-volatile RAM
sysNvRamSet() - write to non-volatile RAM
sysIntDisable() - disable a bus interrupt level
sysIntEnable() - enable a bus interrupt level
sysBusIntAck() - acknowledge a bus interrupt
sysBusIntGen() - generate a bus interrupt
sysMailboxConnect() - connect a routine to the mailbox interrupt
sysMailboxEnable() - enable the mailbox interrupt
sysVicBlkEnable() - initialize the DMA copy operation for the VIC-64.
sysVicBlkAdj() - adjust burst length and interleave period for DMA transfers
sysVicBlkCopy() - copy blocks over the VMEbus using the VIC's DMA feature
sysVicShow() - displays the contents of the VIC-64 registers
sysClkConnect() - connect a routine to the system clock interrupt
sysClkInt() - handle a system clock interrupt
sysClkDisable() - turn off system clock interrupts
sysClkEnable() - turn on system clock interrupts
sysClkRateGet() - get the system clock rate
sysClkRateSet() - set the system clock rate
sysAuxClkConnect() - connect a routine to the auxiliary clock interrupt
sysAuxClkDisable() - turn off auxiliary clock interrupts
sysAuxClkEnable() - turn on auxiliary clock interrupts
sysAuxClkRateGet() - get the auxiliary clock rate
sysAuxClkRateSet() - set the auxiliary clock rate
sysModel() - return the model name of the CPU board
sysBspRev() - return the bsp version with the revision eg 1.0/<x>
sysHwInit() - initialize the CPU board hardware
sysFrontPanelSwitch() - read the front panel switches
sysSetLed() - set value in LED
sysMemTop() - get the address of the top of memory
sysToMonitor() - transfer control to the ROM monitor
sysLocalToBusAdrs() - convert a local address to a bus address
sysBusToLocalAdrs() - convert a bus address to a local address
sysProcNumGet() - get the processor number
sysProcNumSet() - set the processor number
sysBusTas() - test and set a location across the bus
sysSysfailConnect() - connect a routine to the SYSFAIL signal
sysAcfailConnect() - connect a routine to the ACFAIL signal
sysIlaccIntEnable() - enable the ILACC interrupt level

```
STATUS sysNvRamGet
    (char *string, int strLen, int offset)
STATUS sysNvRamSet
    (char *string, int strLen, int offset)
STATUS sysIntDisable
    (int intLevel)
STATUS sysIntEnable
    (int intLevel)
int sysBusIntAck
    (int intLevel)
STATUS sysBusIntGen
    (int level, int vector)
STATUS sysMailboxConnect
    (FUNCPTR routine, int arg)
STATUS sysMailboxEnable
    (char *mailboxAdrs)
STATUS sysVicBlkEnable
    (VOIDFUNCPTR * vector, int level, int burstLength,
     int interleave)
STATUS sysVicBlkAdj
    (int burstLength, int interleave)
STATUS sysVicBlkCopy
    (char * localAddr, char * remoteAddr, int nbytes,
     BOOL toVme)
intsysVicShow()
STATUS sysClkConnect
    (FUNCPTR routine, int arg)
void sysClkInt
    (void)
void sysClkDisable
    (void)
void sysClkEnable
    (void)
int sysClkRateGet
    (void)
STATUS sysClkRateSet
    (int ticksPerSecond)
STATUS sysAuxClkConnect
    (FUNCPTR routine, int arg)
void sysAuxClkDisable
    (void)
void sysAuxClkEnable
    (void)
int sysAuxClkRateGet
    (void)
STATUS sysAuxClkRateSet
    (int ticksPerSecond)
char *sysModel
    (void)
char * sysBspRev
    (void)
void sysHwInit
```

```
(void)
int sysFrontPanelSwitch
    (int swNum)
int sysSetLed
    (int hexVal)
char *sysMemTop
    (void)
STATUS sysToMonitor
    (int startType)
STATUS sysLocalToBusAdrs
    (int adrsSpace, char *localAdrs, char **pBusAdrs)
STATUS sysBusToLocalAdrs
    (int adrsSpace, char *busAdrs, char **pLocalAdrs)
int sysProcNumGet
    (void)
void sysProcNumSet
    (int procNum)
BOOL sysBusTas
    (char *adrs)
STATUS sysSysfailConnect
    (VOIDFUNCPTR routine, int arg)
STATUS sysAcfailConnect
    (VOIDFUNCPTR routine, int arg)
int sysIlaccIntEnable
    (int level)
```

DESCRIPTION

This library provides board-specific routines. The chip drivers included are:

nvRam.c	- non-volatile RAM library
cd2400Serial.c	- Cirrus CD-2400 serial device module library
z8536Timer.c	- Zilog 8536 Counter/Timer device library
vic064Vme.c	- Cypress VIC 068 VMEbus interface controller library

INCLUDE FILES

sysLib.h

SEE ALSO

Programmer's Guide: Configuration

NAME

tyCoDrv - Eltec EUROCOM-17 Cirrus Logic CD2400 MPCC tty driver

SYNOPSIS

tyCoDrv() - tty driver initialization routine
tyCoDevCreate() - create a device for an on-board serial port

```
STATUS tyCoDrv  
    (void)  
STATUS tyCoDevCreate  
    (char * name, int channel, int rdBufSize,  
     int wrtBufSize)
```

DESCRIPTION

This is the driver for the Cirrus Logic CD2400 MPCC. It uses the SCC's in asynchronous mode only.

USER-CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. Two routines, however, must be called directly: tyCoDrv() to initialize the driver, and tyCoDevCreate() to create devices.

Before the driver can be used, it must be initialized by calling tyCoDrv(). This routine should be called exactly once, before any reads, writes, or calls to tyCoDevCreate(). Normally, it is called from usrRoot() in usrConfig.c.

Before a terminal can be used, it must be created using tyCoDevCreate(). Each port to be used should have exactly one device associated with it by calling this routine.

IOCTL FUNCTIONS

This driver responds to the same ioctl() codes as a normal tty driver; for more information, see the manual entry for tyLib. The available baud rates are: 50, 110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200, and 38400.

SEE ALSO

tyLib

NAME

if_il - AMD 79900 ILACC Ethernet network interface driver

SYNOPSIS

ilattach() - publish the interface, and initialize the driver and device

STATUS ilattach

(int unit, char *devAdrs, int ivec, int ilevel,
char *memAdrs, ULONG memSize, int memWidth, int spare,
int spare2)

DESCRIPTION

GENERAL INFORMATION

This module implements the AMD 79900 ILACC Ethernet network interface driver.

This driver is designed to achieve a moderate level of genericism, across the range of architectures and targets that VxWorks 5.1 supports. This allows the driver to be run, unmodified, on a variety of target hardware. To achieve this goal, the driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below. If any of the assumptions mentioned below are not true for your particular hardware, this driver will probably not function correctly. This driver supports only one ILACC unit per CPU. The driver can be configured to support big-endian or little-endian architectures.

BOARD LAYOUT

This device is onboard. No jumpering diagram required.

EXTERNAL INTERFACE

This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach() routine and there is no separate init() routine. Therefore, in the global interface structure, the function pointer to the init() routine is NULL.

There is one user-callable routine: ilattach(). See the manual entry for this routine for usage details.

TARGET SPECIFIC PARAMETERS

buss mode

- This parameter is globally accessed.

The ILACC control register #3 determines the buss mode of the device. This allows the support of big-endian and little-endian architectures. This parameter,

defined as "u_short ilCSR_3B", is the value that will be placed into ILACC control register #3. The default value supports Motorola type busses. See the ILACC manual to change this parameter.

base address of device registers

- This parameter is passed to the driver with the `ilat_tach()` routine.

The ILACC presents two registers to the external interface, the RDP register, and the RAP register. This driver assumes that these two registers occupy two unique addresses in a memory space that is directly accessible by the CPU executing this driver. This driver assumes that the RDP register is mapped at a lower address than the RAP register, and is therefore considered the "base address".

This parameter indicates to the driver where to find the RDP register.

interrupt vector

- This parameter is passed to the driver with the `ilat_tach()` routine.

This driver configures the ILACC device to generate hardware interrupts for various events within the device. Therefore, this driver contains an interrupt handler routine. This driver will call the VxWorks system function `intConnect()` to connect its interrupt handler to the interrupt vector generated as a result of the ILACC interrupt.

interrupt level

- This parameter is passed to the driver with the `ilat_tach()` routine.

Some target hardware use additional interrupt controller devices to help organize and service the various interrupt sources. This driver avoids all board-specific knowledge of such devices. During the initialization of this driver, an external routine is called to perform any board-specific operations required to allow the servicing of a ILACC interrupt. This routine is described below. This parameter is passed to the external routine.

shared memory address

- This parameter is passed to the driver with the `ilat-tach()` routine.

The ILACC device is a DMA type of device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the ILACC. This driver assumes that this shared memory is directly available to this driver without any arbitration or timing concerns.

This parameter may be used to specify an explicit memory region for use by the ILACC. This should be used on hardware that restricts the ILACC to a particular memory region. The constant `NONE` may be used to indicate that there are no memory limitations. In this case, the driver will attempt to allocate the shared memory from the system space.

shared memory size

- This parameter is passed to the driver with the `ilat-tach()` routine.

This parameter can be used to explicitly limit the amount of shared memory (bytes) this driver will use. The constant `NONE` may be used to indicate no specific size limitation. This parameter is only used if a specific memory region is being provided to the driver.

shared memory width

- This parameter is passed to the driver with the `ilat-tach()` routine.

Some target hardware that restricts the shared memory region to a specific location, also restricts the access width to this region by the CPU. On these targets, performing an access of an invalid width will cause a buss error.

This parameter can be used to specify the number of bytes of access width to be used by the driver during access to the shared memory. The constant `NONE` may be used to indicate no restrictions.

The current internal implementation of supporting this mechanism is not robust, and may not work on all targets requiring these restrictions.

shared memory buffer size

- This parameter is passed to the driver with the `ilattach()` routine.

The driver and ILACC device exchange network data in buffers. This parameter allows you to limit the size of these individual buffers. A value of zero indicates that the default buffer size should be used. The default buffer size is large enough to hold a maximum size Ethernet packet.

Use of this parameter should be extremely rare. Network performance will be affected, since the target will no longer be able to receive all legal sizes of packets.

Ethernet address

- This parameter is obtained directly from a global memory location.

During initialization, the driver needs to know the Ethernet address for the ILACC device. The driver assumes that this address is available in a global, 6 byte, character array, named `ilEnetAddr[]`. This array is typically created and stuffed by the BSP code.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires one external support routine to be provided.

`void sysIlaccIntEnable (int level)`

- This routine provides a target-specific enable of the interrupt for the ILACC device. This typically involves interrupt controller hardware, either internal or external to the CPU. This routine is called once, from the `ilattach()` routine.

SYSTEM RESOURCE USAGE

The driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector
- 5016 bytes in code section (text)
- 28 bytes in the initialized data section (data)
- 2244 bytes in the uninitialized data section (bss)

The sections are quoted for the MC68040 architecture and will vary for other architectures.

If the driver allocates the memory to share with the ILACC, it does so by calling the `cacheDmaMalloc()` routine. The

size requested is 80,542 bytes. If a memory region is provided to the driver, the size of this region is adjustable to suit your needs.

The ILACC can only be operated if the shared memory region is write-coherent with regards to the data cache. The driver cannot maintain cache coherency for the device for data that is written by the driver. This is because fields within the shared structures are asynchronously modified by both the driver, and the device, and these fields may share the same cache line.

SEE ALSO
ifLib

NAME

ncr720Lib - NCR 53C720 SCSI I/O Processor (SIOP) library

SYNOPSIS

ncr720CtrlCreate() - create a control structure for the SIOP
ncr720CtrlInit() - initialize a control structure for the
SIOP
ncr720SetHwRegister() - set Hardware dependant registers
ncr720Show() - Display values of all readable ncr720 (SIOP)
registers

```
NCR_720 SCSI_CTRL *ncr720CtrlCreate
    (UINT8 *baseAdrs, UINT freqValue)
STATUS ncr720CtrlInit
    (NCR_720 SCSI_CTRL *pSiop, int scsiCtrlBusId,
     int scsiPriority)
STATUS ncr720SetHwRegister
    (SIOP *pSiop, NCR720_HW_REGS *pHwRegs)
STATUS ncr720Show
    (SCSI_CTRL *pScsiCtrl)
```

DESCRIPTION

This is the I/O driver for the NCR 53C720 SCSI I/O Processor (SIOP). It is designed to work in conjunction with scsiLib. This driver runs in conjunction with a script program for the NCR 53C720 chip. The script uses the NCR 53C720 DMA function for data transfers. This driver supports cache functions through cacheLib.

USER CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. Three routines, however, must be called directly: ncr720CtrlCreate() to create a controller structure, and ncr720CtrlInit() to initialize it. The NCR 53C720 hardware registers need to be configured according to the hardware implementation. If the default configuration is not proper the routine ncr720SetHwRegister() should be used to properly configure the registers.

INCLUDE FILES

ncr720.h, ncr720Script.h

SEE ALSO

scsiLib **Programmer's Guide: I/O System**

NAME

ilattach() - publish the interface, and initialize the driver and device

SYNOPSIS

```
STATUS ilattach
(
    int    unit,          /* unit number */
    char   *devAdrs,      /* ILACC i/o address */
    int    ivec,          /* interrupt vector */
    int    ilevel,        /* interrupt level */
    char   *memAdrs,      /* address of memory pool (-1 == malloc it) */
    ULONG  memSize,       /* only used if memory pool is NOT malloced */
    int    memWidth,      /* byte-width of data (-1 == any width) */
    int    spare,         /* not used */
    int    spare2         /* not used */
)
```

DESCRIPTION

The routine publishes the "il" interface by filling in a network interface record and adding this record to the system list. This routine also initializes the driver and the device to the operational state.

The <memAdrs> parameter can be used to specify the location of the memory that will be shared between the driver and the device. The value NONE may be used to indicate that the driver should obtain the memory.

The <memSize> parameter is only valid if the <memAdrs> parameter is not set to NONE. In this case, this parameter indicates the size of the provided memory region.

The <memWidth> parameter sets the memory pool's data port width (in bytes); if NONE, any data width will be used.

RETURNS

OK or ERROR.

SEE ALSO

if_il

NAME

ncr720CtrlCreate() - create a control structure for the SIOP

SYNOPSIS

```
NCR_720_SCSI_CTRL *ncr720CtrlCreate
(
    UINT8  *baseAdrs, /* base address of the SIOP */
    UINT   freqValue  /* clock controller          */
)
```

DESCRIPTION

This routine creates an SIOP data structure and must be called before using an SIOP chip. It should be called only once for a given SIOP. Since it allocates memory for a structure needed by all routines in ncr720Lib, it must be called before any other routines in the library. After calling this routine, at least one call to ncr720CtrlInit() should be performed before any SCSI transactions are initiated using the SIOP.

A detailed description of the input parameters follows:

<baseAdrs>

- The address at which the CPU would access the lowest register of the SIOP.

<freqValue>

- The value at the SIOP SCSI CLK input. This is used to determine the clock period for the scsi core of the chip and the synchronous divider value for synchronous transfer. It is important to have the right timing on the SCSI bus.

RETURNS

A pointer to NCR_720_SCSI_CTRL structure, or NULL if memory is unavailable or there are bad parameters.

SEE ALSO

ncr720Lib

NAME

ncr720CtrlInit() - initialize a control structure for the SIOP

SYNOPSIS

```

STATUS ncr720CtrlInit
(
    NCR_720_SCSI_CTRL *pSiop,          /* ptr to SIOP
struct                */
    int                scsiCtrlBusId,   /* SCSI bus ID of this
SIOP                  */
    int                scsiPriority     /* priority of a task when doing
SCSI I/O */
)

```

DESCRIPTION

This routine initializes an SIOP structure, after the structure is created with ncr720CtrlCreate(). This structure must be initialized before the SIOP can be used. It may be called more than once if needed; however, it should only be called while there is no activity on the SCSI interface. Before using the SIOP, it must be initialized by calling this routine.

Before returning, this routine pulses RST (reset) on the SCSI bus, thus resetting all attached devices.

A detailed description of the input parameters follows:

<pSiop>

- a pointer to the NCR_720_SCSI_CTRL structure created with ncr720CtrlCreate().

<scsiCtrlBusId>

- the SCSI bus ID of the SIOP. Its value is somewhat arbitrary: seven (7), or highest priority, is conventional. The value must be in the range 0 - 7.

<scsiPriority>

- the priority to which a task is set when performing a SCSI transaction. Legal priorities are 0 to 255. Alternately, a -1 indicates that the priority should not be altered during SCSI transactions.

RETURNS

OK, or ERROR if parameters are out of range.

SEE ALSO

ncr720Lib

NAME

ncr720SetHwRegister() - set Hardware dependant registers

SYNOPSIS

```
STATUS ncr720SetHwRegister
(
    SIOP          *pSiop,    /* Pointer to SIOP infos          */
    NCR720_HW_REGS *pHwRegs /* Pointer to a NCR720_HW_REGS info */
)
```

DESCRIPTION

This routine is used to set up the registers involved in the hardware implementation of the chip. Typically, this routine is called by the sysScsiInit() routine from the bsp. The input parameters are the pointer returned by ncr720CtrlCreate() and a pointer to a NCR720_HW_REGS structure that is filled with the logical values 0 or 1 for each bit of each register describe bellow. This routine includes only the bit registers that could modify the behavior of the chip. The default configuration used during ncr720CtrlCreate(), and ncr720CtrlInit() is {0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0}

```
typedef struct
{
    int ctest4Bit7;    /* Host bus multiplex mode */
    int ctest0Bit7;    /* Disable/enable burst cache capability */
    int ctest0Bit6;    /* Snoop control bit1 */
    int ctest0Bit5;    /* Snoop control bit0 */
    int ctest0Bit1;    /* invert ttl pin (sync bus host mode only) */
    int ctest2Bit5;    /* enable differential scsi bus capability */
    int ctest3Bit0;    /* Set snoop pins mode */
    int dmodeBit7;     /* Burst Length transfer bit 1 */
    int dmodeBit6;     /* Burst Length transfer bit 0 */
    int dmodeBit5;     /* Function code bit FC2 */
    int dmodeBit4;     /* Function code bit FC1 */
    int dmodeBit3;     /* Program data bit (FC0) */
    int dmodeBit1;     /* user programmable transfer type */
    int dcntlBit5;     /* Enable Ack pin */
    int dcntlBit1;     /* Enable fast arbitration on host port */
} NCR720_HW_REGS;
```

To get a more detailed explanation regarding the description of each register involved see the User's Manual of the NCR 53C720.

NOTE

Because this routine writes to the chip registers you can't use it if there is any SCSI bus activity.

ncr720SetHwRegister(2) VXWORKS REFERENCE MANUAnLcr720SetHwRegister(2)

RETURNS

OK or ERROR if any input parameter is NULL

SEE ALSO

ncr720Lib, ncr720.h, ncr720CtrlrCreate()

NAME

ncr720Show() - Display values of all readable ncr720 (SIOP) registers

SYNOPSIS

```
STATUS ncr720Show
(
    SCSI_CTRL *pScsiCtrl /* ptr to SCSI controller info */
)
```

DESCRIPTION

Displays the state of the SIOP registers in a user-friendly way. Primarily used during debugging. The input parameter is the pointer to the SIOP info structure returned by the ncr720CtrlCreate() call.

NOTE

The only readable register during a script execution is Istat register. If you use this routine during the execution of a SCSI comand the result will be unpredictable.

EXAMPLE

```
-> ncr720Show
NCR720 Registers
-----
0xffff47000: Scntl3 = 0xa5 Scntl2 = 0x00 Scntl1 = 0x00 Scntl0 = 0x04
0xffff47004: Gpreg = 0x00 Ssid = 0x00 Sxfer = 0x80 Scid = 0x80
0xffff47008: Sbcl = 0x00 Ssid = 0x00 Socl = 0x00 Sfbr = 0x00
0xffff4700c: Sstat2 = 0x00 Sstat1 = 0x00 Sstat0 = 0x00 Dstat = 0x80
0xffff47010: Dsa = 0x00000000
0xffff47014: Istat = 0x00
0xffff47018: Ctest3 = 0x00 Ctest2 = 0x21 Ctest1 = 0xf0 Ctest0 = 0x00
0xffff4701c: Temp = 0x00000000
0xffff47020: Ctest6 = 0x00 Ctest5 = 0x00 Ctest4 = 0x00 Dfifo = 0x00
0xffff47024: Dcmd/Ddc= 0x50000000
0xffff47028: Dnad = 0x00066144
0xffff4702c: Dsp = 0x00066144
0xffff47030: Dsps = 0x00066174
0xffff47034: Scratch3= 0x00 Scratch2= 0x00 Scratch1= 0x00 Scratch0= 0x0a
0xffff47038: Dcntl = 0x21 Dwt = 0x00 Dien = 0x37 Dmode = 0x01
0xffff4703c: Adder = 0x000cc2b8
0xffff47040: Sist1 = 0x00 Sist1 = 0x00 Sien1 = 0x00 Sien0 = 0x00
0xffff47046: Gpcntl = 0x00 Macntl = 0x00 Swide = 0x00 Slpar = 0x00
0xffff4704a: RespID = Stime1 = 0x00 Stime0 = 0x00
0xffff4704c: Stest3 = 0x00 Stest2 = 0x00 Stest1 = 0x00 Stest0 = 0x00
0xffff47052: Sidl = 0x0000
0xffff47056: Sodl = 0x0000
0xffff4705a: Ssbd1 = 0x0000
0xffff4705c: ScracthB= 0x00000000
value = 0 = 0x0
```

ncr720Show(2)

VXWORKS REFERENCE MANUAL

ncr720Show(2)

SEE ALSO

ncr720Lib, ncr720CtrlCreate()

RETURNS

OK, or ERROR if <pScsiCtrl> and <pSysScsiCtrl> are both NULL.

NAME

sysAcfailConnect() - connect a routine to the ACFAIL signal

SYNOPSIS

```
STATUS sysAcfailConnect
(
    VOIDFUNCPTR routine, /* routine to be connected to ACFAIL signal */
    int          arg      /* argument with which to call routine */
)
```

DESCRIPTION

This routine connects a specified routine to the board's ACFAIL signal.

RETURNS

OK, or ERROR if the routine cannot be connected to the interrupt.

SEE ALSO

sysLib, intConnect()

NAME

sysAuxClkConnect() - connect a routine to the auxiliary clock interrupt

SYNOPSIS

```
STATUS sysAuxClkConnect
(
    FUNCPtr routine, /* routine called at each aux clock interrupt */
    int arg          /* argument with which to call routine */
)
```

DESCRIPTION

This routine specifies the interrupt service routine to be called at each auxiliary clock interrupt. It also connects the clock error interrupt service routine.

RETURNS

OK, or ERROR if the routine cannot be connected to the interrupt.

SEE ALSO

sysLib, intConnect(), sysAuxClkEnable()

NAME

sysAuxClkDisable() - turn off auxiliary clock interrupts

SYNOPSIS

void sysAuxClkDisable (void)

DESCRIPTION

This routine disables auxiliary clock interrupts.

RETURNS

N/A

SEE ALSO

sysLib, sysAuxClkEnable()

NAME

sysAuxClkEnable() - turn on auxiliary clock interrupts

SYNOPSIS

void sysAuxClkEnable (void)

DESCRIPTION

This routine enables auxiliary clock interrupts.

RETURNS

N/A

SEE ALSO

sysLib, sysAuxClkDisable()

NAME

sysAuxClkRateGet() - get the auxiliary clock rate

SYNOPSIS

int sysAuxClkRateGet (void)

DESCRIPTION

This routine returns the interrupt rate of the auxiliary clock.

RETURNS

The number of ticks per second of the auxiliary clock.

SEE ALSO

sysLib, sysAuxClkEnable(), sysAuxClkRateSet()

NAME

sysAuxClkRateSet() - set the auxiliary clock rate

SYNOPSIS

```
STATUS sysAuxClkRateSet
(
    int    ticksPerSecond    /* number of clock interrupts per second */
)
```

DESCRIPTION

This routine sets the interrupt rate of the auxiliary clock. If the auxiliary clock is currently enabled, the clock is disabled and then re-enabled with the new rate.

RETURNS

OK or ERROR.

SEE ALSO

sysLib, sysAuxClkEnable(), sysAuxClkRateGet()

NAME

sysBspRev() - return the bsp version with the revision eg
1.0/<x>

SYNOPSIS

char * sysBspRev (void)

DESCRIPTION

This function returns a pointer to a bsp version with the revision. for eg. 1.0/<x>. BSP_REV defined in config.h is concatenanted to BSP_VERSION defined in bspVersion.h and returned.

RETURNS

A pointer to the BSP version/revision string.

SEE ALSO

sysLib

NAME

sysBusIntAck() - acknowledge a bus interrupt

SYNOPSIS

```
int sysBusIntAck
(
    int  intLevel /* interrupt level to acknowledge */
)
```

DESCRIPTION

This routine acknowledges a specified VMEbus interrupt level. The VIC-64 performs this function automatically.

RETURNS

NULL. Performed by hardware

SEE ALSO

sysLib, sysBusIntGen()

NAME

sysBusIntGen() - generate a bus interrupt

SYNOPSIS

```
STATUS sysBusIntGen
(
    int  level, /* VMEbus interrupt level to generate (1-7) */
    int  vector /* interrupt vector to generate (0-255)      */
)
```

DESCRIPTION

This routine generates a bus interrupt for a specified level with a specified vector.

RETURNS

OK, ERROR if interrupt level out of range (1 - 7) or vector is out of range (0 - 255) or a previous unacknowledged interrupt is pending.

SEE ALSO

sysLib, sysBusIntAck()

NAME

sysBusTas() - test and set a location across the bus

SYNOPSIS

```
BOOL sysBusTas
(
    char *adrs /* address to be tested and set */
)
```

DESCRIPTION

This routine performs a 680x0 test-and-set instruction across the backplane.

RETURNS

TRUE (successful set), or FALSE (failure).

SEE ALSO

sysLib, vxTas()

NAME

sysBusToLocalAdrs() - convert a bus address to a local address

SYNOPSIS

```
STATUS sysBusToLocalAdrs
(
    int    adrsSpace,    /* bus address space in which busAdrs resides */
    char   *busAdrs,     /* bus address to convert                      */
    char   **pLocalAdrs /* where to return local address              */
)
```

DESCRIPTION

This routine gets the local address that accesses a specified VMEbus address.

RETURNS

OK, or ERROR if the address space is unknown or the mapping is not possible.

SEE ALSO

sysLib, sysLocalToBusAdrs()

NAME

sysClkConnect() - connect a routine to the system clock interrupt

SYNOPSIS

```
STATUS sysClkConnect
(
    FUNCPTR routine, /* routine called at each system clock interrupt */
    int      arg      /* argument with which to call routine */
)
```

DESCRIPTION

This routine specifies the interrupt service routine to be called at each clock interrupt. It is called from usrRoot() in usrConfig.c to connect usrClock() to the system clock interrupt. It also connects the clock error interrupt service routine.

RETURNS

OK, or ERROR if the routine cannot be connected to the interrupt.

SEE ALSO

sysLib, intConnect(), usrClock(), sysClkEnable()

NAME

sysClkDisable() - turn off system clock interrupts

SYNOPSIS

void sysClkDisable (void)

DESCRIPTION

This routine disables system clock interrupts.

RETURNS

N/A

SEE ALSO

sysLib, sysClkEnable()

NAME

sysClkEnable() - turn on system clock interrupts

SYNOPSIS

void sysClkEnable (void)

DESCRIPTION

This routine enables system clock interrupts.

RETURNS

N/A

SEE ALSO

sysLib, sysClkConnect(), sysClkDisable(), sysClkRateSet()

NAME

sysClkInt() - handle a system clock interrupt

SYNOPSIS

void sysClkInt (void)

DESCRIPTION

This routine handles a system clock interrupt. It acknowledges the interrupt and calls the routine installed by sysClkConnect().

SEE ALSO

sysLib

NAME

sysClkRateGet() - get the system clock rate

SYNOPSIS

```
int sysClkRateGet (void)
```

DESCRIPTION

This routine returns the interrupt rate of the system clock.

RETURNS

The number of ticks per second of the system clock.

SEE ALSO

sysLib, sysClkEnable(), sysClkRateSet()

NAME

sysClkRateSet() - set the system clock rate

SYNOPSIS

```
STATUS sysClkRateSet
(
    int  ticksPerSecond /* number of clock interrupts per second */
)
```

DESCRIPTION

This routine sets the interrupt rate of the system clock. The new interrupt rate is saved. If the system clock is currently enabled, the clock is disabled and then re-enabled with the new rate. This routine is called by usrRoot() in usrConfig.c.

NOTE

The valid range for the system clock is 40 to 5000 ticks per second.

RETURNS

OK, or ERROR if the tick rate is invalid or the timer cannot be set.

SEE ALSO

sysLib, sysClkEnable(), sysClkRateGet()

NAME

sysHwInit() - initialize the CPU board hardware

SYNOPSIS

void sysHwInit (void)

DESCRIPTION

This routine initializes various features of the CPU board hardware. It is called from usrInit() in usrConfig.c.

NOTE

This routine should not be called by the user.

RETURNS

N/A

SEE ALSO

sysLib

NAME

sysIlaccIntEnable() - enable the ILACC interrupt level

SYNOPSIS

```
int sysIlaccIntEnable
(
    int level /* interrupt level, not used */
)
```

DESCRIPTION

This routine enables interrupts for the on-board ILACC chip at a specified level. ILACC interrupts are controlled by the VIC chip. The VIC chip also provides the interrupt vector.

RETURNS

OK

SEE ALSO

sysLib

NAME

sysIntDisable() - disable a bus interrupt level

SYNOPSIS

```
STATUS sysIntDisable
(
    int  intLevel /* interrupt level to disable (1-7) */
)
```

DESCRIPTION

This routine disables a specified VMEbus interrupt level.

RETURNS

OK, or ERROR if <intLevel> is not in the range 1 - 7.

SEE ALSO

sysLib, sysIntEnable()

NAME

sysIntEnable() - enable a bus interrupt level

SYNOPSIS

```
STATUS sysIntEnable
(
    int    intLevel /* interrupt level to enable (1-7) */
)
```

DESCRIPTION

This routine enables a specified VMEbus interrupt level.

RETURNS

OK, or ERROR if <intLevel> is not in the range 1 - 7.

SEE ALSO

sysLib, sysIntDisable()

NAME

sysLocalToBusAdrs() - convert a local address to a bus address

SYNOPSIS

```
STATUS sysLocalToBusAdrs
(
    int    adrsSpace,    /* bus address space in which busAdrs resides */
    char   *localAdrs,   /* local address to convert */
    char   **pBusAdrs    /* where to return bus address */
)
```

DESCRIPTION

This routine gets the VMEbus address that accesses a specified local memory address.

RETURNS

OK, or ERROR if the mapping is not possible.

SEE ALSO

sysLib, sysBusToLocalAdrs()

NAME

sysMailboxConnect() - connect a routine to the mailbox interrupt

SYNOPSIS

```
STATUS sysMailboxConnect
(
    FUNCPTR routine, /* routine called at each mailbox interrupt */
    int      arg      /* argument with which to call routine      */
)
```

DESCRIPTION

This routine specifies the interrupt service routine to be called at each mailbox interrupt.

RETURNS

OK, always

SEE ALSO

sysLib, intConnect(), sysMailboxEnable()

NAME

sysMailboxEnable() - enable the mailbox interrupt

SYNOPSIS

```
STATUS sysMailboxEnable
(
    char *mailboxAdrs /* address of mailbox (ignored) */
)
```

DESCRIPTION

This routine enables the ICMS-0 mailbox on the VIC064.

RETURNS

OK, always.

SEE ALSO

sysLib, sysMailboxConnect()

NAME

sysMemTop() - get the address of the top of memory

SYNOPSIS

char *sysMemTop (void)

DESCRIPTION

This routine finds the size of memory.

RETURNS

The address of the top of memory.

SEE ALSO

sysLib

NAME

sysModel() - return the model name of the CPU board

SYNOPSIS

char *sysModel (void)

DESCRIPTION

This routine returns the model name of the CPU board.

RETURNS

A pointer to the string "Eltec EUROCOM-17".

SEE ALSO

sysLib

NAME

sysNvRamGet() - get the contents of non-volatile RAM

SYNOPSIS

```
STATUS sysNvRamGet
(
    char *string, /* where to copy non-volatile RAM */
    int  strLen,  /* maximum number of bytes to copy */
    int  offset   /* byte offset into non-volatile RAM */
)
```

DESCRIPTION

This routine copies the contents of non-volatile memory into a specified string. The string will be terminated with an EOS.

RETURNS

OK, or ERROR if access is outside the non-volatile RAM range.

SEE ALSO

sysLib, sysNvRamSet()

NAME

sysNvRamSet() - write to non-volatile RAM

SYNOPSIS

```
STATUS sysNvRamSet
(
    char *string, /* string to be copied into non-volatile RAM */
    int  strLen,  /* maximum number of bytes to copy */
    int  offset   /* byte offset into non-volatile RAM */
)
```

DESCRIPTION

This routine copies a specified string into non-volatile RAM.

RETURNS

OK, or ERROR if access is outside the non-volatile RAM range.

SEE ALSO

sysLib, sysNvRamGet()

NAME

sysProcNumGet() - get the processor number

SYNOPSIS

```
int sysProcNumGet (void)
```

DESCRIPTION

This routine returns the processor number for the CPU board, which is set with sysProcNumSet().

RETURNS

The processor number for the CPU board.

SEE ALSO

sysLib, sysProcNumSet()

NAME

sysProcNumSet() - set the processor number

SYNOPSIS

```
void sysProcNumSet
(
    int  procNum
)
```

DESCRIPTION

This routine sets the processor number for the CPU board.
Processor numbers should be unique on a single backplane.

RETURNS

N/A

SEE ALSO

sysLib, sysProcNumGet()

NAME

sysSetLed() - set value in LED

SYNOPSIS

```
int sysSetLed
(
    int  hexVal  /* value to display in LED display */
)
```

DESCRIPTION

This routine sets a value into the hex display. When the value to display is negative, then the display will be blanked.

RETURNS

OK

SEE ALSO

sysLib

NAME

sysSysfailConnect() - connect a routine to the SYSFAIL signal

SYNOPSIS

```
STATUS sysSysfailConnect
(
    VOIDFUNCPTR routine, /* routine to be connected to SYSFAIL signal */
    int arg             /* argument with which to call routine */
)
```

DESCRIPTION

This routine connects a specified routine to the board's SYSFAIL signal.

RETURNS

OK, or ERROR if the routine cannot be connected to the interrupt.

SEE ALSO

sysLib, intConnect()

NAME

sysToMonitor() - transfer control to the ROM monitor

SYNOPSIS

```
STATUS sysToMonitor
(
    int startType /* parameter passed to ROM to tell it how to boot */
)
```

DESCRIPTION

This routine transfers control to the ROM monitor. Normally, it is called only by reboot()--which services ^X--and bus errors at interrupt level. However, in some circumstances, the user may wish to introduce a <startType> to enable special boot ROM facilities.

RETURNS

Does not return.

SEE ALSO

sysLib

NAME

sysVicBlkAdj() - adjust burst length and interleave period
for DMA transfers

SYNOPSIS

```
STATUS sysVicBlkAdj
(
    int  burstLength, /* burst length of DMA block transfer (1-64) */
    int  interleave   /* interleave period between bursts (1-15)   */
)
```

DESCRIPTION

This routine changes the burst length and interleave period that are to be used for VME DMA transfers. <burstLength> is the number of VMEbus cycles per burst, <interleave> is the local cycle interleave period between block transfers. The interleave time is equal to 250ns times <interleave>.

RETURNS

OK, or ERROR if parameters are out of range.

SEE ALSO

sysLib, sysVicBlkEnable(), sysVicBlkCopy()

NAME

sysVicBlkCopy() - copy blocks over the VMEbus using the VIC's DMA feature

SYNOPSIS

```
STATUS sysVicBlkCopy
(
    char * localAddr, /* local address */
    char * remoteAddr, /* vme address */
    int nbytes, /* transfer length in bytes */
    BOOL toVme /* copy local memory to vme (or vise-versa) */
)
```

DESCRIPTION

This routine uses the DMA feature of the VIC to do block-mode copies to/from the VMEbus. This routine takes a local address <localAddr> and remote address <remoteAddr> and transfers <nbytes> bytes in the direction specified by <toVme>.

RETURNS

OK, or ERROR if transfer was unsuccessful. Unsuccessful transfers result from bus errors or bad alignments.

SEE ALSO

sysLib, sysVicBlkEnable(), sysVicBlkAdj()

NAME

sysVicBlkEnable() - initialize the DMA copy operation for the VIC-64.

SYNOPSIS

```

STATUS sysVicBlkEnable
(
    VOIDFUNCPTR * vector,      /* interrupt vector for DMA done interrupt
*/
    int          level,        /* interrupt level for DMA done interrupt
*/
    int          burstLength,  /* burst length of DMA block transfer (1-
64) */
    int          interleave    /* interleave period between bursts (1-
15) */
)

```

DESCRIPTION

This routine initializes operations for DMA transfers using the VIC-64. The DAM done interrupt is attached using the <vector> as the interrupt vector and <level> as the interrupt level. <burstLength> is the number of VMEbus cycles per burst, <interleave> is the local cycle interleave period between block transfers. The interleave time is equal to 250ns times <interleave>.

This routine is typically called by sysHwInit2().

RETURNS

OK or ERROR, if the DMA paramters are not valid.

SEE ALSO

sysLib, sysVicBlkAdj(), sysVicBlkCopy()

NAME

sysVicShow() - displays the contents of the VIC-64 registers

SYNOPSIS

```
int sysVicShow()
```

DESCRIPTION

This routine displays the contents of the VIC-64 registers.

RETURNS

OK, always.

SEE ALSO

sysLib,

NAME

tyCoDevCreate() - create a device for an on-board serial port

SYNOPSIS

```
STATUS tyCoDevCreate
(
    char *   name,          /* name to use for this device */
    int      channel,       /* physical channel for this device */
    int      rdBufSize,     /* read buffer size, in bytes */
    int      wrtBufSize     /* write buffer size, in bytes */
)
```

DESCRIPTION

This routine creates a device on a specified serial port. Each port to be used should have exactly one device associated with it by calling this routine.

For instance, to create the device "/tyCo/0", with buffer sizes of 512 bytes, the proper call would be:

```
tyCoDevCreate ("/tyCo/0", 0, 512, 512);
```

RETURNS

OK, or ERROR if the driver is not installed or the channel is invalid.

SEE ALSO

tyCoDrv